Project documentation

ELEC-A7151 Dungeon Crawler Project

Jonas Edström 713892, Henrik Niskanen 709178
Oskar Björkgren 705868, Mikael Virkkunen 770644

1.Overview

In this project we have made a dungeon crawler game. The object of the game is to defeat the monsters and finally beat the boss. The game is won once the boss is eliminated. Battles between players and monsters are carried out by shooting projectiles. The game is played in real time and can be paused by pressing the ESC key. The player is moved by pressing and holding the WASD keys and projectiles can be shot using the mouse (LEFT CLICK). The speed of the player can be increased using the LSHIFT, making the player dash for a limited amount of time. The player is able to move through a randomly created map with rooms, where each room has a slightly different background. The spawn room can easily be identified as it has no extra vegetation. There are always 10 rooms in the game, which could easily be changed by tweaking the room generator. The way the rooms are connected varies from game to game but the room generator ensures that all the rooms are connected and that it is possible to reach the boss. The rooms contain monsters of a randomly chosen type. All the monsters in a room have to be eliminated before a room is cleared, else the monsters will reappear in the room once the player re-enters.

There are a total of 5 different monster types in the game with different movement and attacking capabilities.

- The random monster moves in a random direction for a certain amount of time and then changes its direction. The random monster shoots projectiles aimed at the player.
- The wall patrol monster patrols the walls of the room. It shoots projectiles aimed at the player.
- The sniping monster cannot move but has a much longer range. It shoots projectiles aimed at the player.
- The searching monster moves toward the player. It does not shoot any projectiles but directly damages the player when it gets close enough.
- The slow monster moves toward the player. It shoots projectiles with a very fast fire rate but it’s aim is not that great. It also is as the name says, very slow.

As the monsters are eliminated there is a possibility that a potion is dropped at the position of the monster. Every monster drops a potion with equal probability and the type of potion is also independent of the monster type. The player can pick up the potions by getting sufficiently close to the potion and the potion is then added to the player’s inventory. The number potions in the player’s inventory is displayed in the top right corner. The potions can later be used by pressing the number keys 1,2,3 or 4 corresponding to using a green, red, yellow and violet potion respectively.

The current status of the game is displayed in the upper left and right corners. The green bar at the top left indicates the health of the player. Once the health of the player is 0, the player dies and the player is prompted to restart the game. The violet bar indicates the cooldown for attacking once the bar is filled the player is able to attack. As the player performs an attack the width of the bar goes to zero and starts increasing until the player is able to attack again. The yellow bar indicates the corresponding functionality for dashing.

The player's inventory is shown in the upper right corner. The green (5hp) and red (10hp) potions increase the health of the player, the yellow potions give the player the ability to use dash for a longer time and the violet potions increase the damage made by the player's attacks for the upcoming three attacks. The potions are not carried over when the player dies.

In the unfortunate case that the player dies, it is possible to restart the game and the player will retain some of the attributes from the previous attempt like the level and XP. Thus, the game can be played continuously until the players abilities have been increased so much that it becomes virtually impossible to not win the game.

2.Software Structure

The structure of our project is largely based on inheriting classes and virtual functions.

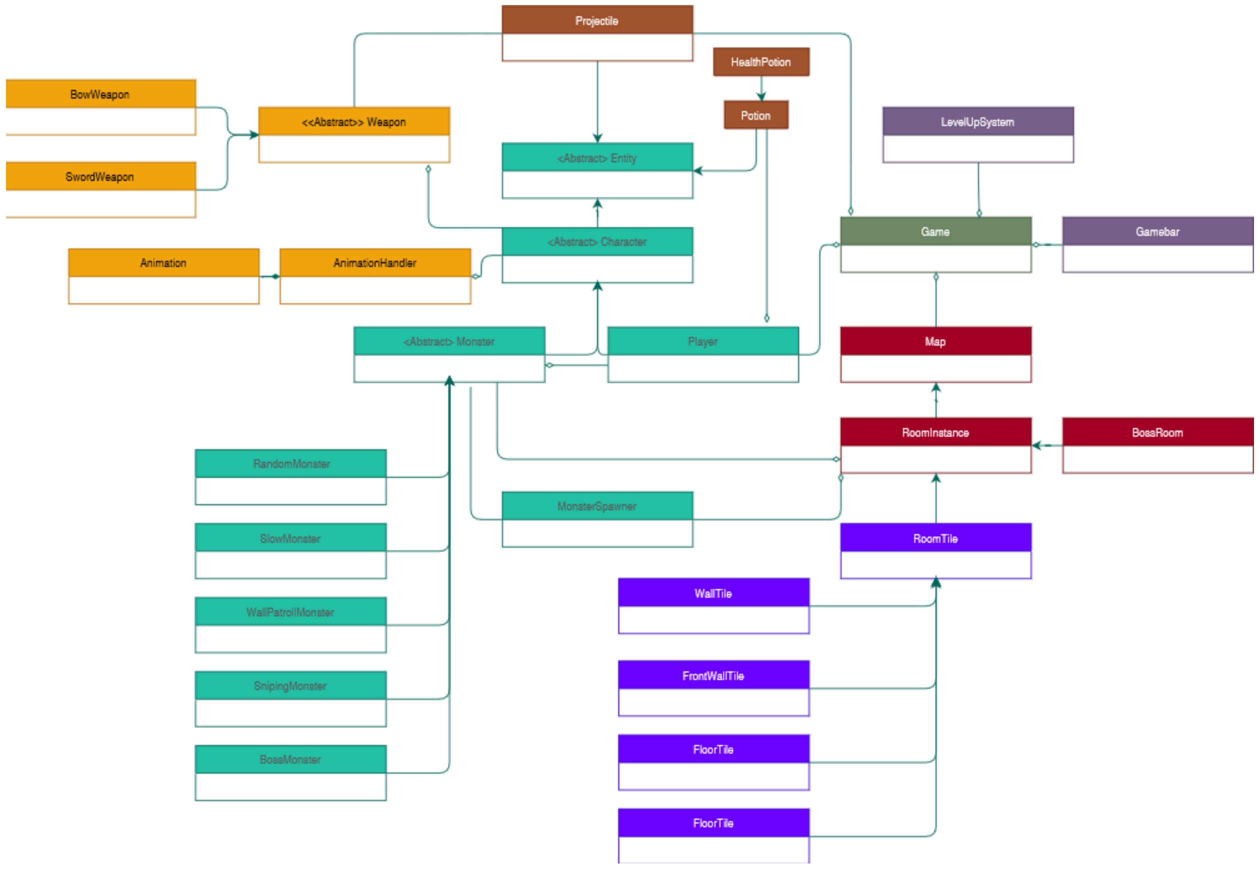
The base thing in our game is the entity class. The idea of the entity class is that almost everything needs some basic variables and methods. For example a position and some getters and setters.

The character class is a subclass of the entity class. All moving "beings" are subclasses of the character class. It has some methods that are common between all "beings".

We have some different monsters that are all subclasses of a Monster class. The player class is a subclass of the character class.

We have some utility methods that are implemented under the randomhelper, spritehelper and the animationhandler. They are used to deal with some general essentials that a lot of different classes need. For example generating random ints and floats and dealing with sprites and animations.

Generally we have a game class which has a map, player and gamebar. The game is responsible for updating everything that needs to be updated. The map has a map containing all roomInstances in our game. Each room is responsible for keeping track of the monsters and potions that are inside them. Under is a more detailed UML diagram of our class structure with a link to a draw.io project.



https://drive.google.com/file/d/1Wqv9G_xwMBR0OV1qz6wXgH8vJG8nP8Jq/view

3.Instructions for building and using software

We based the building of the software on andre-r-king's sfml-vscode-boilerplate, which makes building the software very easy. We realized too late that its linking is dependent on a local SFML-installation and modifying it proved to be a more difficult task than expected, after countless hours of tweaking and testing we decided that we'll let it be with minor tweaks.

How to run:
As it is still dependent on the local installation for linking we have to first install SFML if not installed. On (debian based) linux distros the command is:

```
> sudo apt-get install libsfml-dev
```

In VS Code select a build & run, there is debug, release, production and tests.
Tests naturally runs the tests in the /tests-folder.
Alternatively one can run it from the terminal with a the command:

```
> make run
```

Which builds and runs the game in production mode.

4.Testing

Due to a lack of time the number of "real" tests in the project is very limited. However, we managed to implement the Catch2 framework for automated testing. The Catch2 framework was used to create some basic unit tests for the entity, character and player classes. Most of the "real" testing was done by actually playing the game and trying to find bugs with different user inputs. A lot of testing was also carried out using caveman-esque print functions.
Testing was used mostly for detecting memory-leaks, which there are still a few of, the SFML-library also seemed to always produce some memory leaks with textures.

5.Work log

We have had some quite separate responsibilities when developing the project.

Henrik has been the architectural mastermind. He has approved and read through most of the merge requests so that we avoid merging bad or messy code. Also did a massive amount of work into making the project build better.

Mikael has been responsible for developing some of the essential classes and features like entity, projectile and the dash ability. Also work on some algorithms for generating the dungeon. Done a lot of stuff that just someone needs to do.

Jonas has made a lot of the UI and made some of the essential classes and features like monsters and healing.

Oskar has been in charge of animations and visuals. He has made a lot of our sprites that we use and single handedly made the animations.

Here is a summary of what our group has done each week. One can find some more details and more about our work in the meeting notes and the git commits/merges.

Week 1 (1.11-7.11)

The first week was really about setting everything up to then being able to start working on the programming of our project. We also made a UML class diagram about how we thought that our classes would interact. Some of our group members had not used git before so we also educated ourselves on how to use it. A lot of time also went to writing the project plan. In the project plan we decided on some preliminary responsibilities for each project member.
Oskar: Game design and Visuals
Henrik: Software architecture
Jonas: Characters
Mikael: Game Cycle and movement

We did manage to stick to these responsibilities surprisingly well.

Week 2 (8.11-14.11)

During the second week we developed our project from just having set up SFML to having keyboard input, projectiles, player and more. Since none of us really had any experience on SFML or c++ game programming, quite a lot of time went to research before we could start writing code and developing our project. All of our group members did research into how SFML works and about typical class relations and functions in already existing SFML games on the internet. We did know that we were going to need a character class and projectiles and we found out that it is highly recommended to compensate for different frame rates with a deltaTime variable.

Mikael implemented deltaTime (dt_) so that our project will compensate for different computers having varying frame rates when running our project. This works by passing the dt_ variable into all functions that move entities. When the frame rate is high the dt_ variable will be low and the movement per game cycle will be short. If we did not compensate for this entities would move faster when the game is played on a better computer. Mikael also started work on the projectile class and a basic collision system. We were going to use sprite boundaries to check if a projectile was colliding with a character. This all took around 8h of work.

During the second week Jonas started working on the character class. We thought that it would be logical to first implement the character class and then write the subclasses (player and monsters).

Henrik had many other things to do and didn't have much time to do development. He set up some labels for git and looked into how to structure the game, he also had git crash courses for the others. ~9h.

Oskar set up a framework for the game loop, worked on getting an animated walk cycle for the player character. Also worked on the game design part of things, feel, look of the game and mechanics. Started also looking into the implementation of rooms. ~ 10hours.

Week 3 (15.11-21.11)

During the third week we managed to implement a lot of some basic but very crucial game features. We continued work on the projectiles and how they collide with what kind of characters. We found a pixel perfect collision file on the internet that we decided that we wanted to use. We wanted to give some more feedback to the player so we decided to implement a gamebar which will display the hitpoints of the player. A weapon was added to the game that allowed characters to shoot projectiles. The first room was developed but the player was still able to walk through its walls. During this week we also had to do the first bigger merges between branches.

Mikael continued to work on the projectiles. Added a projectile type that we used to only allow Monsters to shoot the Player and vice versa. A function to check collisions between projectiles and other sprites was implemented. When the projectile collides with some other sprite it gets deleted. If the projectile hits a character the game checks if the projectile type is allowed to hit that character and then deals damage to the character. Also implemented the pixel perfect collision system found on the internet to work with our game. This all took around 14h of work.

Jonas: Created the first monster classes. Still had a lot to learn about how git should be used and tried to keep up with other people’s code. Trying to figure out how the classes should interact also took some time. The total workload was around 12 hours.

Henrik started making an inventory system which was later abandoned when the design of the game changed. He then started making weapons that spawn projectiles that Mikael had implemented and could be equipped to characters. He also reviewed a lot of code and made decisions about using another clang format for the project. ~5h

Oskar worked on implementing the room instances from last week based on a tile/grid format. Started working on wall collision and trying to figure out how to generate the dungeon. About 7h of work.

Week 4 (22.11-28.11)

During the fourth week we implemented some more optional and advanced features but also continued on working on the must have essentials. This week was also a cleanup week. We did a lot of housekeeping. We cleaned up and redesigned some of the already written code. We developed the first monster types. Some research was done into cooldown systems so we could make a cooldown for the attacks and implement a dash feature. We also started thinking about doxygen and how we would document our project.

Mikael did a lot of research into how cooldowns for attacks and other features are usually done in SFML. The decision was made that passing the deltaTime variable into functions that implement these features was the best way for our project. When this was decided the attack could be placed under this cooldown system. Then Mikael developed a dash function using the same idea. This all took around 10h of work.

Jonas: Created the bars showing the hitpoints of the player and the monsters. Created some simple attack functions for the monsters. In total around 8 hours was spent on the project.

Henrik continued his endeavors concerning weapons and ended up reworking a lot of the projectile logic, added custom sprites and made sprites align themselves in the direction they had been shot. He and Mikael worked on making a roadmap of finishing the project and set up git issues for the features. Did some housekeeping in the project. Added collision detection and handling for movement and figured out how doxygen works and how to use it in VS code. ~20h

Oskar got started with actually implementing the map, trying to make it possible to walk from room to room. Started work on some small parts like power ups that were abandoned. Made some changes to sprites and added more animation to the player like an idle function. ~12h

Week 5 (29.11-5.12)

The fifth week the game began to be playable. Some bugs were fixed. We finally made it possible to walk from room to room and the first version of the dungeon generation was developed. The attack methods needed a big rework. The gamebar was developed further to show the remaining cooldowns for dashing and attacking. A level system was developed to allow the player and other characters to gain XP and get stronger and better. The monster spawner was developed alongside three new monster types. The map needed work on and we made it so that it now consists of different tiles, so that the rooms could be different from others.

Mikael reworked the attack functions of the different character subclasses to be easier to use and make more sense. This was because our game supports two completely different ways of dealing damage. The first is shooting projectiles and the other is directly decreasing the hitpoints of the character being attacked. The player should only be able to use attacks based on projectiles. Some monsters use projectile attacks while some directly decrease hitpoints. This took a lot of time to rework the code so it supported these attacks on the correct characters and weapons. Then he developed a system to level up different characters. Our game only levels up the player but the level up system supports leveling up of all characters in a clean and efficient way. The level of the character can then be used to buff the hitpoints and attack damage. He also made three more monster types with some unique attacks and behavior. This all took about 18h of work.

Jonas: Did not contribute with much code but rather spent time on reading up on other members code and tried to get a better overview of the project. Total time spent on the project 4h.

Henrik did some improvements to the text of the “gamebar”, fixed aim working also when the game window is resized, did some improvements to the game engine, e.g. added abstractions and extracted functions. He created a monster spawner that spawns monsters in the game’s roominstances. He also took over the map generation branch from Oskar as he was a bit opinionated about how it should and should not be done. 25h

Oskar continued working on the map and rooms, like adding a spawn room. Got stuck pretty hard with implementation and Henrik took over the map generation. Worked on making the room look different by creating new tiles and randomizing the generation. Started work on weapon classes with plans to make weapons you could be picked up(abandoned because of a lack of time). Sword weapon which also got too finicky to finish implementing. ~5/6h

Week 6 (6.12-12.12)

As usually with projects a lot of features and different todos were started and completed in this last week. We wrote some tests to check the correctness of some of our methods. The dungeon generation was redeveloped so it cannot not fail and be less buggy in general. The collision functions were made polymorphic and a lot more readable. A lot of effort was put towards trying to improve the building of the game and making it not dependant on the user having SFML installed locally. The death functionality and winning conditions and the bossmonster were implemented. A system for picking up and using potions to heal and in other ways buff the player was made. Some basic sound effects to give the player better feedback when projectiles are hitting something where added. Also this project documentation was written during this week.

Mikael redeveloped the dungeon generation. The version before this had some issues where the map generation could fail. In this new generation system the generation is now both quick and should not be able to fail. The pause functionality was made so that the player can pause the game if he/her needs a break. The boss monster along with some unique attacks was developed. He also wrote doxygen documentation for a lot of methods that he previously had written. Tweaked some variables and fixed bugs. For example the player could shoot when dead and the gamebar had some visual bugs that could be solved with min and max functions. Implemented the sound effects for when projectiles collide with characters. This was originally developed by Oskar but now implemented in the master. Since Mikael also was the only member with a computer running Linux he had to test the sound effects. This all took about 40h of work.

Jonas: Added all the potions to the game and created an inventory for the potions. Also tried to help Henrik fixing the valgrind errors and wrote some text. Added some tests at the end but the tests were very basic. Tried to help other group members wherever possible and started writing the documentation. Total time spent on the project during the week: 25h.

Henrik integrated the monster spawner to the map, made QOL fixes to the game. He pondered a lot about the linking libraries and the makefile but ended up just thinking about SFML without the ‘S’. Fixed memory leaks and used smart pointers. >40h

Oskar finished the win cons for our game, wrote documentation, fixed sprites and projectiles for monsters so they all have a different feel. A lot of patching up bits and pieces. ~25h